

Experiment Number	6
Experiment Title	Experiment 6: Web Scraper using LLMs
Date of Experiment	13/03/2025
Date of Submission	20/03/2025

Experiment 6: Web Scraper using LLMs

Objective:

To create a web scraper application integrated with LLMs for processing scraped data.

Detailed Procedure:

- 1. Use Python libraries like BeautifulSoup and Requests to scrape web data.**
- 2. You can also use LlamaIndex for Web Scraping and Ollama for open ended LLMs**
- 2. Integrate LLMs to process and summarize the scraped information.**
- 3. Develop a Flask backend for handling scraping tasks and queries.**
- 4. Create an HTML/CSS frontend to initiate scraping (like the web page to scrape) and display results.**
- 5. You can also take a topic and search the web for a web page and then scrape it.**

Code:- From next page as follows

app.py, llama_processor.py, scraper.py, search.py, index.html, style.css, script.js

```

1 from flask import Flask, render_template, request, jsonify
2 from modules.scrapers import WebScraper
3 from modules.llm_processor import LLMProcessor
4 from modules.search import WebSearch
5
6 app = Flask(__name__)
7
8 # Initialize modules
9 scraper = WebScraper()
10 llm = LLMProcessor()
11 search = WebSearch()
12
13 @app.route('/')
14 def index():
15     return render_template('index.html')
16
17 @app.route('/scrape', methods=['POST'])
18 def scrape_url():
19     data = request.json
20     url = data.get('url')
21     scrape_method = data.get('method', 'bs4') # Default to BeautifulSoup
22
23     if not url:
24         return jsonify({"error": "URL is required"}), 400
25
26     try:
27         # Scrape content based on the selected method
28         if scrape_method == 'llamaindex':
29             result = scraper.scrape_with_llamaindex(url)
30         else:
31             result = scraper.scrape_with_bs4(url)
32
33         # Check if scraping was successful
34         if not result.get('content'):
35             return jsonify({"error": result.get('error', 'Failed to scrape content')}), 400
36
37         return jsonify(result)
38
39     except Exception as e:
40         return jsonify({"error": str(e)}), 500
41
42 @app.route('/process', methods=['POST'])
43 def process_content():
44     data = request.json
45     content = data.get('content')
46     processing_type = data.get('type', 'summarize') # Default to summarize
47
48     if not content:
49         return jsonify({"error": "Content is required"}), 400
50
51     try:
52         # Process content based on the selected processing type
53         if processing_type == 'keypoints':
54             result = llm.extract_key_points(content)
55         elif processing_type == 'sentiment':
56             result = llm.analyze_sentiment(content)
57         else:
58             result = llm.summarize(content)
59
60         return jsonify({"result": result})
61
62     except Exception as e:
63         return jsonify({"error": str(e)}), 500
64
65 @app.route('/search', methods=['POST'])
66 def search_topic():
67     data = request.json
68     topic = data.get('topic')
69
70     if not topic:
71         return jsonify({"error": "Topic is required"}), 400
72
73     try:
74         # Get URLs for the topic
75         urls = search.get_urls_for_topic(topic)
76
77         if not urls:
78             return jsonify({"error": "No URLs found for the topic"}), 404
79
80         return jsonify({"urls": urls})
81
82     except Exception as e:
83         return jsonify({"error": str(e)}), 500
84
85 @app.route('/scrape-and-process', methods=['POST'])
86 def scrape_and_process():
87     data = request.json
88     url = data.get('url')
89     topic = data.get('topic')
90     processing_type = data.get('processingtype', 'summarize')
91
92     # Either URL or topic must be provided
93     if not url and not topic:
94         return jsonify({"error": "Either URL or topic is required"}), 400
95
96     try:
97         # If topic is provided but URL is not, search for URLs
98         if topic and not url:
99             urls = search.get_urls_for_topic(topic)
100             if not urls:
101                 return jsonify({"error": f"No URLs found for topic: {topic}"}), 404
102             url = urls[0] # Use the first URL
103
104         # Scrape the content
105         result = scraper.scrape_with_bs4(url)
106
107         if not result.get('content'):
108             return jsonify({"error": result.get('error', 'Failed to scrape content')}), 400
109
110         # Process the content based on the selected type
111         if processing_type == 'keypoints':
112             processed_result = llm.extract_key_points(result['content'])
113         elif processing_type == 'sentiment':
114             processed_result = llm.analyze_sentiment(result['content'])
115         else:
116             processed_result = llm.summarize(result['content'])
117
118         return jsonify({
119             'url': url,
120             'title': result.get('title', 'No title'),
121             'domain': result.get('domain', ''),
122             'scraped_content': result['content'][:500] + "...", # Preview of content
123             'processed_result': processed_result
124         })
125
126     except Exception as e:
127         return jsonify({"error": str(e)}), 500
128
129 if __name__ == '__main__':
130     app.run(debug=True)
131

```

```

1 import requests
2
3 class LLMProcessor:
4     def __init__(self, model="llama2"):
5         """Initialize the LLM processor with the specified model"""
6         self.model = model
7         self.ollama_url = "http://localhost:11434/api"
8         self._ensure_model_available()
9
10    def _ensure_model_available(self):
11        """Check if the model is available and pull it if needed"""
12        try:
13            # Check if Ollama is running
14            response = requests.get(f"{self.ollama_url}/tags")
15
16            # Check if our model is in the list
17            models = response.json().get("models", [])
18            model_exists = any(m["name"] == self.model for m in models)
19
20            if not model_exists:
21                print(f"Model {self.model} not found. Pulling from Ollama...")
22                self._pull_model()
23
24        except requests.RequestException:
25            print("Ollama service not detected. Please ensure Ollama is installed and running.")
26            print("Visit https://ollama.ai/ for installation instructions.")
27
28    def _pull_model(self):
29        """Pull the model from Ollama"""
30        try:
31            response = requests.post(
32                f"{self.ollama_url}/pull",
33                json={"name": self.model}
34            )
35
36            if response.status_code == 200:
37                print(f"Successfully pulled {self.model}")
38            else:
39                print(f"Failed to pull {self.model}: {response.text}")
40        except Exception as e:
41            print(f"Error pulling model: {str(e)}")
42
43    def process_content(self, content, instruction):
44        """Process content with the LLM based on the given instruction"""
45        try:
46            prompt = f"""
47            {instruction}
48
49            Content to process:
50            {content[:10000]} # Limiting to first 10K chars to avoid token limits
51            """
52
53            response = requests.post(
54                f"{self.ollama_url}/generate",
55                json={
56                    "model": self.model,
57                    "prompt": prompt,
58                    "stream": False
59                }
60            )
61
62            if response.status_code == 200:
63                result = response.json()
64                return result.get("response", "No response generated")
65            else:
66                return f"Error: {response.status_code} - {response.text}"
67        except Exception as e:
68            return f"Error processing content: {str(e)}"
69
70    def summarize(self, content):
71        """Summarize the provided content"""
72        instruction = "Please provide a concise summary of the following content:"
73        return self.process_content(content, instruction)
74
75    def extract_key_points(self, content):
76        """Extract key points from the provided content"""
77        instruction = "Please extract the main points and key information from the following content:"
78        return self.process_content(content, instruction)
79
80    def analyze_sentiment(self, content):
81        """Analyze the sentiment of the provided content"""
82        instruction = "Please analyze the sentiment of the following content. Is it positive, negative, or neutral? Explain your reasoning:"
83        return self.process_content(content, instruction)
84
85

```

```

1 import requests
2 from bs4 import BeautifulSoup
3 from llama_index.readers.web import TrafilaturaWebReader
4 import re
5 from urllib.parse import urlparse
6
7 class WebScraper:
8     def __init__(self):
9         self.session = requests.Session()
10        self.session.headers.update({
11            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
12        })
13
14    def scrape_with_bs4(self, url):
15        """Scrape web content using BeautifulSoup"""
16        try:
17            response = self.session.get(url, timeout=10)
18            response.raise_for_status()
19
20            soup = BeautifulSoup(response.content, 'html.parser')
21
22            # Remove script and style elements
23            for script in soup(["script", "style"]):
24                script.extract()
25
26            # Get text content
27            text = soup.get_text(separator='\n')
28
29            # Clean up text: remove extra newlines and whitespace
30            clean_text = re.sub(r'\n+', '\n', text).strip()
31            clean_text = re.sub(r'\s+', ' ', clean_text)
32
33            # Extract title
34            title = soup.title.string if soup.title else "No title found"
35
36            # Extract domain
37            domain = urlparse(url).netloc
38
39            return {
40                "url": url,
41                "title": title,
42                "domain": domain,
43                "content": clean_text
44            }
45        except Exception as e:
46            return {
47                "url": url,
48                "error": str(e),
49                "content": None
50            }
51
52    def scrape_with_llamaindex(self, url):
53        """Scrape web content using LlamaIndex's SimpleWebPageReader"""
54        try:
55            SimpleWebPageReader = download_loader("SimpleWebPageReader")
56            loader = SimpleWebPageReader()
57            documents = loader.load_data(urls=[url])
58
59            if documents and len(documents) > 0:
60                return {
61                    "url": url,
62                    "title": documents[0].metadata.get("title", "No title found"),
63                    "domain": urlparse(url).netloc,
64                    "content": documents[0].text
65                }
66            else:
67                return {
68                    "url": url,
69                    "error": "No content found",
70                    "content": None
71                }
72        except Exception as e:
73            return {
74                "url": url,
75                "error": str(e),
76                "content": None
77            }
78

```

```

1 import requests
2 from bs4 import BeautifulSoup
3 from urllib.parse import quote_plus
4
5 class WebSearch:
6     def __init__(self):
7         self.session = requests.Session()
8         self.session.headers.update({
9             'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
10        })
11
12    def search_google(self, query, num_results=5):
13        """
14        Search Google for the query and return a list of result URLs.
15        This is a basic implementation and might be blocked by Google.
16        For production, consider using a proper search API.
17        """
18        try:
19            search_url = f"https://www.google.com/search?q={quote_plus(query)}&num={num_results}"
20            response = self.session.get(search_url, timeout=10)
21            response.raise_for_status()
22
23            soup = BeautifulSoup(response.text, 'html.parser')
24
25            # Find all result links
26            search_results = []
27            for result in soup.select('div.g div.yuRUbf a'):
28                url = result['href']
29                if url.startswith('http') and 'google.com' not in url:
30                    search_results.append(url)
31
32            return search_results[:num_results]
33
34        except Exception as e:
35            print(f"Error searching: {str(e)}")
36            return []
37
38    def get_urls_for_topic(self, topic, num_results=5):
39        """Get a list of relevant URLs for a given topic"""
40        return self.search_google(topic, num_results)
41

```

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Web Scraper with LLM</title>
7   <link rel="icon" type="image/x-icon" href="{{ url_for('static', filename='favicon.ico') }}">
8   <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
9 </head>
10 <body>
11   <div class="container">
12     <header>
13       <h1>ScrapWorld</h1>
14       <p>Scrape web content and process it with large language models</p>
15     </header>
16
17     <div class="tabs">
18       <button class="tab-btn active" data-tab="url-tab">Scrape by URL</button>
19       <button class="tab-btn" data-tab="topic-tab">Scrape by Topic</button>
20     </div>
21
22     <div class="tab-content active" id="url-tab">
23       <div class="input-group">
24         <label for="url-input">URL to Scrape:</label>
25         <input type="text" id="url-input" placeholder="https://example.com">
26       </div>
27
28       <div class="options-group">
29         <label>Scraping Method:</label>
30         <div class="radio-group">
31           <input type="radio" id="bs4-method" name="scrape-method" value="bs4" checked>
32           <label for="bs4-method">BeautifulSoup</label>
33
34           <input type="radio" id="llamaindex-method" name="scrape-method" value="llamaindex">
35           <label for="llamaindex-method">LlamaIndex</label>
36         </div>
37       </div>
38
39       <button id="scrape-url-btn" class="primary-btn">Scrape</button>
40     </div>
41
42     <div class="tab-content" id="topic-tab">
43       <div class="input-group">
44         <label for="topic-input">Topic to Search:</label>
45         <input type="text" id="topic-input" placeholder="Enter a topic...">
46       </div>
47       <button id="search-topic-btn" class="primary-btn">Search & Scrape</button>
48     </div>
49
50     <div class="processing-options">
51       <label>Processing Type:</label>
52       <div class="radio-group">
53         <input type="radio" id="summarize" name="processing-type" value="summarize" checked>
54         <label for="summarize">Summarize</label>
55
56         <input type="radio" id="keypoints" name="processing-type" value="keypoints">
57         <label for="keypoints">Extract Key Points</label>
58
59         <input type="radio" id="sentiment" name="processing-type" value="sentiment">
60         <label for="sentiment">Analyze Sentiment</label>
61       </div>
62     </div>
63
64     <div class="results-container" id="results-container" style="display: none;">
65       <div class="loading" id="loading">
66         <div class="spinner"></div>
67         <p>Processing... This may take a moment</p>
68       </div>
69
70       <div class="results" id="results">
71         <div class="result-meta">
72           <h2 id="result-title">Title</h2>
73           <p id="result-url">URL: <a href="#" target="_blank"></a></p>
74         </div>
75
76         <div class="result-tabs">
77           <button class="result-tab-btn active" data-result-tab="processed">Processed Content</button>
78           <button class="result-tab-btn" data-result-tab="raw">Raw Content</button>
79         </div>
80
81         <div class="result-tab-content active" id="processed-content">
82           <pre id="processed-result"></pre>
83         </div>
84
85         <div class="result-tab-content" id="raw-content">
86           <pre id="raw-result"></pre>
87         </div>
88       </div>
89     </div>

```

```

1  /* Base styles */
2  * {
3    box-sizing: border-box;
4    margin: 0;
5    padding: 0;
6  }
7
8  body {
9    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
10   line-height: 1.6;
11   color: #333;
12   background-color: #f5f5f5;
13 }
14
15 .container {
16   max-width: 800px;
17   margin: 0 auto;
18   padding: 20px;
19 }
20
21 /* Header */
22 header {
23   text-align: center;
24   margin-bottom: 30px;
25 }
26
27 header h1 {
28   color: #2c3e50;
29   margin-bottom: 10px;
30 }
31
32 header p {
33   color: #7f8c8d;
34 }
35
36 /* Tabs */
37 .tabs {
38   display: flex;
39   margin-bottom: 20px;
40   border-bottom: 1px solid #ddd;
41 }
42
43 .tab-btn {
44   padding: 10px 20px;
45   border: none;
46   background: none;
47   cursor: pointer;
48   font-size: 16px;
49   color: #7f8c8d;
50   transition: all 0.3s ease;
51 }
52
53 .tab-btn.active {
54   color: #3498db;
55   border-bottom: 2px solid #3498db;
56 }
57
58 .tab-content {
59   display: none;
60   margin-bottom: 20px;
61 }
62
63 .tab-content.active {
64   display: block;
65 }
66
67 /* Form elements */
68 .input-group {
69   margin-bottom: 15px;
70 }
71
72 .input-group label {
73   display: block;
74   margin-bottom: 5px;
75   font-weight: 500;
76 }
77
78 .input-group input {
79   width: 100%;
80   padding: 10px;
81   border: 1px solid #ddd;
82   border-radius: 4px;
83   font-size: 16px;
84 }
85
86 .options-group {
87   margin-bottom: 15px;
88 }
89
90 .radio-group {
91   display: flex;
92   gap: 15px;
93   margin-top: 5px;
94 }
95
96 /* Buttons */
97 .primary-btn {
98   background-color: #3498db;
99   color: white;
100  border: none;
101  padding: 10px 20px;
102  font-size: 16px;
103  border-radius: 4px;
104  cursor: pointer;
105  transition: background-color 0.3s ease;
106 }
107
108 .primary-btn:hover {
109   background-color: #2980b9;
110 }
111
112 /* Processing options */
113 .processing-options {
114   margin-bottom: 20px;
115   padding: 15px;
116   background-color: #ecf0f1;
117   border-radius: 4px;
118 }
119
120 .processing-options label {
121   font-weight: 500;
122   margin-bottom: 10px;
123   display: block;
124 }
125
126 /* Results container */
127 .results-container {
128   margin-top: 10px;
129   border: 1px solid #ddd;
130   border-radius: 8px;
131   overflow: hidden;
132 }
133
134 .loading {
135   display: flex;
136   flex-direction: column;
137   align-items: center;
138   justify-content: center;
139   padding: 40px;
140 }
141
142 .spinner {
143   border: 4px solid rgba(0, 0, 0, 0.1);
144   border-left-color: #3498db;
145   border-radius: 50%;
146   width: 40px;
147   height: 40px;
148   animation: spin 1s linear infinite;
149   margin-bottom: 15px;
150 }
151
152 @keyframes spin {
153   0% { transform: rotate(0deg); }
154   100% { transform: rotate(360deg); }
155 }
156
157 .results {
158   display: none;

```

```

1 document.addEventListener('DOMContentLoaded', function() {
2     // Tab switching
3     const tabBtns = document.querySelectorAll('.tab-btn');
4     const tabContents = document.querySelectorAll('.tab-content');
5
6     tabBtns.forEach(btn => {
7         btn.addEventListener('click', () => {
8             // Deactivate all tabs
9             tabBtns.forEach(b => b.classList.remove('active'));
10            tabContents.forEach(c => c.classList.remove('active'));
11
12            // Activate selected tab
13            btn.classList.add('active');
14            const tabId = btn.getAttribute('data-tab');
15            document.getElementById(tabId).classList.add('active');
16        });
17    });
18
19    // Result tab switching
20    const resultTabBtns = document.querySelectorAll('.result-tab-btn');
21    const resultTabContents = document.querySelectorAll('.result-tab-content');
22
23    resultTabBtns.forEach(btn => {
24        btn.addEventListener('click', () => {
25            // Deactivate all result tabs
26            resultTabBtns.forEach(b => b.classList.remove('active'));
27            resultTabContents.forEach(c => c.classList.remove('active'));
28
29            // Activate selected result tab
30            btn.classList.add('active');
31            const tabId = btn.getAttribute('data-result-tab') + '-content';
32            document.getElementById(tabId).classList.add('active');
33        });
34    });
35
36    // Scrape URL button click handler
37    document.getElementById('scrape-url-btn').addEventListener('click', () => {
38        const url = document.getElementById('url-input').value.trim();
39        if (!url) {
40            alert('Please enter a URL to scrape');
41            return;
42        }
43
44        const scrapeMethod = document.querySelector('input[name="scrape-method"]:checked').value;
45        const processingType = document.querySelector('input[name="processing-type"]:checked').value;
46
47        // Show loading state
48        showLoading();
49
50        // Make API call to scrape and process
51        fetch('/scrape-and-process', {
52            method: 'POST',
53            headers: {
54                'Content-Type': 'application/json'
55            },
56            body: JSON.stringify({
57                url: url,
58                scrapeMethod: scrapeMethod,
59                processingType: processingType
60            })
61        })
62        .then(response => response.json())
63        .then(data => {
64            if (data.error) {
65                throw new Error(data.error);
66            }
67            displayResults(data);
68        })
69        .catch(error => {
70            hideLoading();
71            alert('Error: ' + error.message);
72        });
73    });
74
75    // Search topic button click handler
76    document.getElementById('search-topic-btn').addEventListener('click', () => {
77        const topic = document.getElementById('topic-input').value.trim();
78        if (!topic) {
79            alert('Please enter a topic to search');
80            return;
81        }
82
83        const processingType = document.querySelector('input[name="processing-type"]:checked').value;
84
85        // Show loading state
86        showLoading();
87
88        // Make API call to search, scrape and process
89        fetch('/scrape-and-process', {
90            method: 'POST',
91            headers: {
92                'Content-Type': 'application/json'
93            },
94            body: JSON.stringify({
95                topic: topic,
96                processingType: processingType
97            })
98        })
99        .then(response => response.json())
100        .then(data => {
101            if (data.error) {
102                throw new Error(data.error);
103            }
104            displayResults(data);
105        })
106        .catch(error => {
107            hideLoading();
108            alert('Error: ' + error.message);
109        });
110    });
111
112    // Function to display results
113    function displayResults(data) {
114        // Hide loading indicator
115        hideLoading();
116
117        // Display results container
118        document.getElementById('results-container').style.display = 'block';
119
120        // Fill in result data
121        document.getElementById('result-title').textContent = data.title || 'No Title';
122        const urlElement = document.querySelector('#result-url a');
123        urlElement.textContent = data.url;
124        urlElement.href = data.url;
125    }

```


4. Results/Output:- Entire Screen Shot including Date & Time

The image displays two screenshots of the ScrapWorld web application interface, which is used for scraping web content and processing it with large language models.

Top Screenshot (Scrape by Topic):

- ScrapWorld** logo and tagline: "Scrape web content and process it with large language models".
- Navigation tabs: "Scrape by URL" and "Scrape by Topic" (selected).
- Topic to Search:** A text input field with the placeholder "Enter a topic...".
- Search & Scrape** button.
- Processing Type:** Radio buttons for "Summarize" (selected), "Extract Key Points", and "Analyze Sentiment".

Bottom Screenshot (Scrape by URL):

- ScrapWorld** logo and tagline: "Scrape web content and process it with large language models".
- Navigation tabs: "Scrape by URL" (selected) and "Scrape by Topic".
- URL to Scrape:** A text input field containing "https://example.com".
- Scraping Method:** Radio buttons for "BeautifulSoup" (selected) and "LlamaIndex".
- Scrape** button.
- Processing Type:** Radio buttons for "Summarize" (selected), "Extract Key Points", and "Analyze Sentiment".

5. Remarks:-

Signature of the Student

Apratim Dutta
(Name of the Student)

Signature of the Lab Coordinator

(Name of the Coordinator)